VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

JavaScript Renderer

JavaScript Renderer

2013 Belešová Jana

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jana Belešová**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Javascript Renderer

Javascript Renderer

Zásady pro vypracování:

Cílem této bakalářské práce je implementace webového javascript rendereru. Implementace bude konkrétně zahrnovat generování geometrických tvarů kvádr a koule, ořezání zorným objemem, mapování textur a rasterizaci. Student k implementaci využije html5 elementu canvas pro zobrazení výsledné scény. Součástí práce bude měření výkonosti javascriptového engine moderních webových prohlížečů (firefox, internet explorer, chrome) prostřednicvtím vykreslování scén v implementovaném rendereru.

Seznam doporučené odborné literatury:

[1] LUBBERS, Peter, ALBERS, Brian, SALIM, Frank. HTML5: Programujeme moderní webové aplikace. P?el. O. Gibl. 1. vydání. Brno: Computer Press a.s., 2011. 304 s. ISBN 978-80-251-3539-6
[2] MEYER, Jeanine. HTML5 and JavaScript Projects. 1. vydání. Apress. 2011. 448 s. ISBN 978-1430240327
[3] WELLMAN, Dan. jQuery UI 1.6: The User Interface Library for jQuery. Birmingham: Packt Publishing Ltd. 2009. 420 s. ISBN 978-1-847195-12-8
[4] McFARLAND, David Saywer. JavaScript & jQuery: The Missing Manual. 2.vydání. Pogue Press. 2011. 540 s. ISBN 978-1449399023

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Lukáš Zaorálek**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 2. května 2013 ..... Bělšová Jana .....

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 2. května 2013 ..... Bělšová Jana .....

Na tomto místě bych chtěla poděkovat Ing. Lukáši Zaorálkovi za užitečné rady pří tvoření této práce, dále bych chtěla poděkovat všem, kteří mě podporovali.

Abstrakt

Práce se zabývá vykreslováním grafiky ve webovém prohlížeči za použití HTML5 elementu canvas a jazyka JavaScript.

V praktické části byl implementován renderer, který vykresluje objekty kvádr a koule potažené texturou.

V textové části jsou vysvětleny jednotlivé kroky, které k vytvoření rendereru vedly. Také je zde zahrnuto měření výkonnosti webových prohlížečů, pomocí vykreslování scén v implementovaném rendereru.

Klíčová slova: JavaScript, HTML5, canvas, renderování, 3D, počítačová grafika, webové prohlížeče, výkonnost prohlížečů

Abstract

This thesis is about displaying graphics in web browser by using HTML5 element canvas and JavaScript language.

In practical part the renderer has been implemented, which can render cuboid and sphere with texture.

Single steps which lead to create the renderer are explained in theretical part of this thesis. This JavaScript renderer has been used to measure web browser performance. Results of these tests are in theoretical part of this thesis.

Keywords: JavaScript, HTML5, canvas, renderering, 3D, computer graphics, web browsers, browsers performance

Seznam použitých zkratek a symbolů

2D	–	dvourozměrný
3D	–	trojrozměrný
API	–	Application Programming Interface
CPU	–	Central Processing Unit
CSS	–	Cascading Style Sheet
DOM	–	Document Object Model
FPS	–	Frames Per Second
HTML	–	HyperText Markup Language
HTTP	–	(Hypertext Transfer Protocol
IETF	–	Internet Engineering Task Force
ISO	–	International Organization for Standardization
JPEG	–	Joint Photographic Experts Group
JS	–	JavaScript
NDC	–	Normalized Device Coordinates
OS	–	operační systém
RAM	–	Random-access memory
SQL	–	Structured Query Language
SVG	–	Scalable Vector Graphics
VRML	–	Virtual Reality Model Language
W3C	–	World Wide Web Consortium
WebGL	–	Web Graphics Library
WHATWG	–	Web Hypertext Application Technology Working Group
WWW	–	World Wide Web
X3D	–	Extensible 3D
XML	–	Extensible Markup Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

Seznam tabulek

1 Úvod

S rozvojem výpočetní techniky roste význam počítačové grafiky a s čím dál větší dostupností internetového připojení se rozšiřuje používání webových aplikací. S příchodem technologie HTML5 se rozšířili možnosti webových aplikací v mnoha oblastech, i v oblasti počítačové grafiky. V této práci je popsáno, jak je možné pomocí HTML5 elementu `canvas` a čistého JavaScriptu vykreslovat 3D objekty ve webovém prohlížeči.

Popisu zvolených technologií, kterými jsou HTML5 a Javascript, je věnována kapitola 3.

HTML5 není jedinou technologií, která je schopna renderovat grafiku ve webovém prohlížeči, dalšími jsou například Java3D nebo WebGL, které využívají pro reprezentaci 3D dat speciální formáty, kterými jsou VRML nebo X3D. Touto tématikou se zabývám v kapitole 2.

Samotný proces, při kterém dochází k vizualizaci 3D dat na 2D výstupní zařízení, se skládá z mnoha kroků, které zajišťují co nejvěrnější výsledek. Mezi tyto kroky patří samotná reprezentace objektu, jeho transformace. Dále zavedení kamery do scény, která představuje pozorovatele a pomocí perspektivní projekce simuluje vjemy lidského oka. Kroky sloužící k optimalizaci jako ořezání zorným objemem nebo odstranění zadních ploch. Pro reálný vzhled objektů se provádí mapování textur a na konec je potřeba celou scénu vykreslit, čemuž se říká rasterizace. Tento proces se nazývá zobrazovací řetězec a je popsán v kapitole 4.

Kapitola 5 je věnována webovým prohlížečů, je zde popsána obecná architektura webového prohlížeče, jednotlivá renderovací a JavaScriptová jádra. Dále jsou zde prezentovány výsledky měření výkonnosti prohlížečů za použití implementovaného rendereru.

2 Rendering

Při renderování dochází k napodobování a zobrazení reálného světa, reprezentovaného nejčastěji 3D modelem, na obrazovce počítače nebo jiného zařízení. Reálnost výsledného obrazu je ovlivněna mnoha vlastnostmi, které je možné simulovat například zdroj světla, textury nebo mlhu. Cílem této práce je realizování renderování ve webovém prohlížeči za pomocí HTML5 elementu `canvas` za použití jazyka JavaScript. K renderování ve webovém prohlížeči je možné použit i jiné technologie jako například WebGL nebo Java3D.

2.1 3D webové technologie

Pro práci s 3D daty lze využit mnoha různých technologií, pro zobrazení těchto dat je však většinou nutné použít zásuvné moduly. Často se pro popis 3D objektů využívá speciálních formátů, jakými jsou VRML nebo X3D. Tyto formáty jsou dále využívány při vytváření virtuálního světa.

2.1.1 VRML

VRML (Virtual Reality Modeling Language) je textový formát pro popis virtuální scény. Poprvé byl představen v roce 1994 na konferenci WWW. V roce 1997 byla vydána finální verze VRML 2.0 označována také jako VRML97 a stala se standardem ISO. Pro zobrazení scény vytvořené pomocí tohoto formátu je potřeba do webového prohlížeče doinstalovat speciální přehrávač.

Scénu je možné vytvářet z lokálních souborů i souborů umístěných na internetu, které mají příponu .wrl. Při tvorbě scény se využívá stromové struktury, objekty jsou uzly stromu a tvoří graf scény. Uzly mohou obsahovat téměř jakékoli informace např. 3D reprezentaci tělesa nebo textury ve formátu JPEG. Ve scéně je možné kromě jednotlivých objektů a jejich chování definovat také zdroj světla, kameru nebo barevnou mlhu. [11, 3]

```
#VRML V2.0 utf8
# A Cylinder
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Cylinder {
        height 2.0
        radius 1.5
    }
}
```

Výpis 1: Syntaxe VRML 2.0

2.1.2 X3D

X3D (Extensible 3D) je formát pro ukládání 3D scény. Může být reprezentován textově, binárně nebo pomocí XML, soubory jsou ukládány s příponou .x3d. Vychází z VRML 2.0

a textová verze je s VRML kompatibilní, obsahuje však mnohem více funkcí. Stejně jako u VRML je nutné pro zobrazení scény použít některý z doplňků pro prohlížení X3D scén.

X3D stejně jako VRML definuje geometrii a chování objektů scény, ty jsou rozděleny do logických tříd (komponent). Umožňuje s objekty pracovat v jiných souřadnicový soustavách.[3, 8]

```
<Transform>
    <Shape>
        <Appearance>
            <Material diffuseColor="0␣1␣0"/>
        </Appearance>
        <Cylinder height="0.1" radius="0.5"/>
    </Shape>
</Transform>
```

Výpis 2: Syntaxe X3D pomocí XML

2.1.3 Java3D

Java3D je přídavná knihovna v jazyce Java, vytvořená za účelem zobrazování 3D grafiky. Java je multiplatformní jazyk, proto i vizualizace vytvořené pomocí knihovny Java3D mohou být zobrazovány na různých zařízeních a pomocí appletu i v prostředí internetu. Objekty je možné načítat ve formátu VRML.

Applety vytvořené pomoci Java3D je možné integrovat do HTML 4.01 pomocí tagu `<applet>`. HTML5 tento tag již nepodporuje, lze jej nahradit pomocí tagu `<object>`.[9]

```
<applet code="Bubbles.class" width="350" height="350">
Java applet that draws animated bubbles.
</applet>
```

Výpis 3: Použití Java appletu v HTML kódu

2.1.4 WebGL

WebGL je multiplatformní 3D API vycházející z OpenGL ES 2.0 a je spravováno skupinou Khronos. Specifikace WebGL 1.0 byla zveřejněna v roce 2011. Slouží k vytváření 3D grafiky ve webovém prohlížeči bez použití zásuvných modulů, pro zobrazení výstupu využívá HTML5 elementu `canvas` a DOM rozhraní.

K vykreslování do elementu `canvas` se využívá `WebGLRenderingContext`, který umožňuje přístup k funkcím WebGL API. Přístup k tomuto contextu lze získat pomocí příkazu `canvas.getContext('webgl')`. WebGL není podporováno ve všech moderních mobilních prohlížečích, přestože tyto prohlížeče element `canvas` podporují. [10]

3 Použité technologie

3.1 HTML5

HTML5 je značkovací jazyk pro strukturování a prezentaci obsahu v prostředí internetu. Je ve stavu návrhu, konečná verze by měl být uvolněna koncem roku 2014, verze 5.1 pak koncem roku 2016. Za specifikaci jsou odpovědny tři organizace a to WHATWG, W3C a IETF. Konsorcium W3C do roku 2006 pracovalo na XHTML 2.0, poté od tohoto vývoje upustili a začali pracovat na nové verzi HTML společně s WHATWG. IETF je zodpovědná za internetové protokoly a tedy i za WebSocket API, které je součásti specifikace HTML5.

Velkou výhodou HTML5 je podpora mnoha funkcí, které bylo dříve možné použít jen s využitím zásuvných modulů. Žádný z prohlížečů nepodporuje všechny funkce HTML5, ale nejvíce používané prohlížeče podporují mnoho funkcí a podporu rozšiřují v nových verzích. Také je možné použitím metod Canvas API vykreslovat nejrůznější grafické prvky v mobilních prohlížečích, což například s technologii WebGL možné není.

Některými z nových funkcí jsou:

- Canvas
- Geolokace
- MathML
- Microdata
- Webová SQL databáze
- SVG

Kromě nových funkcí jsou v HTML5 také nové nebo upravené tagy (značky). Například specifikace dokumentu se zjednodušila na tvar `<!DOCTYPE html>`, kódování dokumentu je také zjednodušeno `<meta charset="UTF-8">`.

3.1.1 Canvas API

Vykreslování scén pomocí JavaScriptového rendereru, který je v rámci této práce implementován, je realizováno právě za použití elementu `canvas`, proto jej zde krátce popíši.

`Canvas` slouží k vykreslování grafiky ve webovém prohlížeči pomocí jazyka JavaScript. Má tvar obdélníku a do webové stránky jej lze přidat použitím tagu `<canvas>` Pro samotné vykreslování je možné využít mnoha funkcí a lze zobrazovat křivky, rovinné útvary, texty, stíny nebo také obrázky.

Pro práci s jednotlivými metodami elementu `canvas` se využívá jeho `context`, konkrétně `CanvasRenderingContext2D` s nimž lze pracovat pomocí jazyka JavaScript. `Context` elementu `canvas` získáme pomocí příkazu `canvas.getContext('2d')`. [2]

```
<!DOCTYPE html>
<html>
    <head>
    <meta charset="UTF-8">
        <title>
        Vykresleni cerveneho obdelniku.
        </title>
    </head>
    <body>
        <canvas id="canvas" width="800" height="600">
                Vas prohlizec nepodporuje element canvas.
        </canvas>
        <script>
        var canvas = document.getElementById("canvas");
        var ctx = canvas.getContext("2d");

        ctx.fillStyle ="#FF0000";
        ctx.fillRect (0,0,150,75);
        </script>
    </body>
<html>
```

Výpis 4: Použití HTML5 elementu canvas

HTML5 element `canvas` je podporován v prohlížečích - Internet Explorer 9+, Chrome, Opera, Firefox a Safari.

3.2 JavaScript

JavaScript je skriptovací programovací jazyk, který řeší dynamiku webových stránek na straně klienta. Zdrojový kód se zpracovává přímo v prohlížeči. Jedná se i multiplatformní, objektové orientovaný, ale bez třídní jazyk.

Poprvé byl představen v roce 1995 pod názvem LiveScript jako součást webového prohlížeče NetScape Navigator. V roce 1996 uvedl Microsoft svou verzi tohoto jazyka nazývanou JScript, kterou využívá ve svých prohlížečích. Roku 1997 byla vydána standardizace ECMAScript za účelem poskytnutí společného jádra pro všechny prohlížeče.

Do HTML kódu se přidává pomocí tagu `script`, a to buď uvedením zdrojového souboru se skriptem, nebo se mezi tagy umístí přímo kód.

```
<!DOCTYPE html>
<html>
    <head>
        <script>
        function myFunction()
        {
        alert("Hello World!");
        }
        </script>
    </head>

    <body>
```

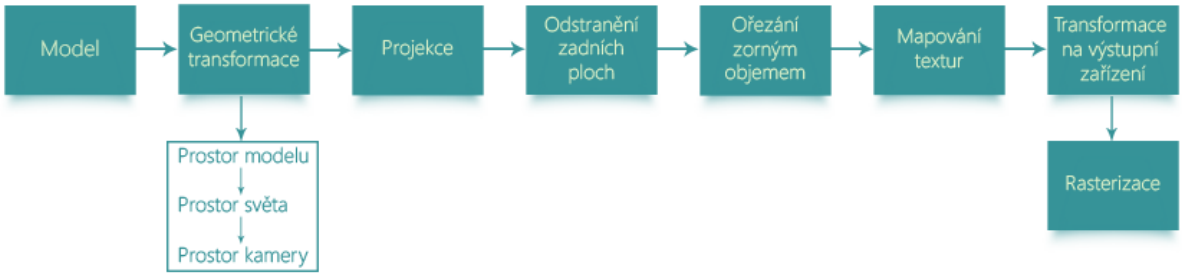
```
    <button onclick="myFunction()">Try it</button>
    </body>
</html>
```

Výpis 5: Ukázka funkce v jazyce JavaScript

JavaScriptový kód je možné ladit pomocí nástrojů ve webových prohlížečích, například Firefox má přídavný nástroj FireBug. Některé prohlížeče mají již zabudované nástroje pro vývojáře.

4 Zobrazovací řetězec

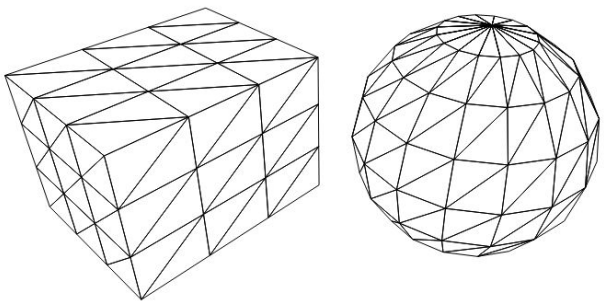
Jako zobrazovací řetězec je nazývána posloupnost kroků, která převede 3D scénu do 2D zobrazení na výstupním zařízení. Kroky v řetězci mohou být různě uspořádány. V případě této práce je používán řetězec na obrázku 4, který obsahuje reprezentaci 3D modelu jeho převedení do prostoru světa a prostoru kamery, toto je realizováno pomocí transformačních matic. Dále je provedena perspektivní projekce, která simuluje reálné optické jevy. Dále je prováděno odstranění zadních ploch, což jsou plochy které jsou od pozorovatele odvráceny. Dalším krokem je ořezání zorným objemem, poté jsou objekty potaženy texturou a vykresleny na výstupní zařízení.



Obrázek 1: Zobrazovací řetězec

4.1 Model

Scéna je tvořena množinou objektů, jejich povrch je popsán pomocí sítě mnohoúhelníků (polygonů), nejjednoduššími jsou trojúhelníky, ty si uchovávají informace o svých vrcholech, které jsou dány kartézskými souřadnicemi (x, y, z). Pomocí trojúhelníků je možné popsat jak rovinné plochy, tak plochy zakřivené. Takový popis 3D objektu se nazývá 3D model. Prostor, v jehož středu se nachází střed modelu, se nazývá **prostor modelu**.



Obrázek 2: 3D model kvádru a koule

4.2 Svět

Prostor světa přestavuje scénu, ve které se nacházejí jednotlivé modely. Protože i v reálném světe mají jednotlivé objekty různá umístění je potřeba definovat tato umístění pro jednotlivé objekty ve scéně a to převedením modelu do **prostoru světa**. Toho dosáhneme aplikováním geometrických transformací, ty jsou představovány transformačními maticemi, které jsou popsány níže. Pro použití těchto transformací, kterými jsou posunutí, změna měřítka a rotace, je třeba zavést homogenní souřadnice.

4.2.1 Homogenní souřadnice

Homogenní souřadnice rozšiřují prostor o jednu dimenzi a umožňují reprezentovat body v nekonečnu (nevlastní body). Zjednodušují maticové operace a umožňují použití matice posunutí. Pomocí těchto souřadnic lze také realizovat středové promítání. Bod $P(x, y, z)$ zapíšeme pomocí homogenních souřadnic jako $P^h(wx, wy, wz, w)$, kde w je libovolné reálné číslo různé od nuly.

4.2.2 Transformační matice v homogenních souřadnicích

Posunutí o t_x na ose X, t_y na ose Y a t_z na ose Z.

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Měřítko

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Rotace kolem osy X o úhel α.

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & sin(\alpha) & 0 \\ 0 & -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Rotace kolem osy Y o úhel α.

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} cos(\alpha) & 0 & -sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ sin(\alpha) & 0 & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Rotace kolem osy Z o úhel α.

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} cos(\alpha) & sin(\alpha) & 0 & 0 \\ -sin(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Vynásobením vrcholů modelu těmito maticemi převedeme model z prostoru modelu do prostoru světa. Je však důležité uvědomit si, co se má s modelem provést a podle toho také volit pořadí násobení matic, násobení matic není komutativní. Pokud nejprve model posuneme a poté otočíme, pak se bude model otáčet podle středu světa. Jestliže chceme, aby se model otáčel kolem své osy, je potřeba nejprve provést otočení a poté až posunutí. Matice se nejčastěji násobí v tomto pořadí: *měřítko * rotace * posunutí*. Avšak pří implementaci může být pořadí opačné, záleží na tom, jestli je prováděno násobení matic zprava nebo zleva. V případě této práce dochází k násobení matic zprava, aby bylo dodrženo správné násobení matic je při implementaci využito pořadí *posunutí * rotace * měřítko*.

4.3 Kamera

Dalším krokem je transformace do **prostoru kamery**, ta je provedena vynásobením pohledové matice maticí kamery. Kamera ve scéně simuluje pozici pozorovatele. Pro vytvoření pohledové matice je potřeba znát pozici kamery $P(p_x, p_y, p_z)$, směr pohledu kamery (target) $T(t_x, t_y, t_z)$ a vektor určující směr vzhůru $Up(up_x, up_y, up_z)$. Pohledová matice se skládá ze tří vektorů N, V, U, které představují osy souřadného systém.

Pohledová matice má tvar:

$$\begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Pohled lidského oka směřuje po ose Z, tu představuje v pohledové matici vektor N, osou Z bude tedy vektor směřující k cíli.

$$N = \frac{T}{|T|}$$

Vektor N a vektor směřující vzhůru Up se nacházejí v jedné rovině, jejich vektorovým součinem získáme vektor U tedy osu X.

$$U = \frac{Up \times N}{|Up \times N|}$$

Vektor Up směřuje vzhůru, ale není zajištěna jeho kolmost s osou X, nelze jej tedy použit pro reprezentaci osy Y, tu získáme jako vektorový součin osy X a Z, tedy vektorů N a U.

$$V = U \times N$$

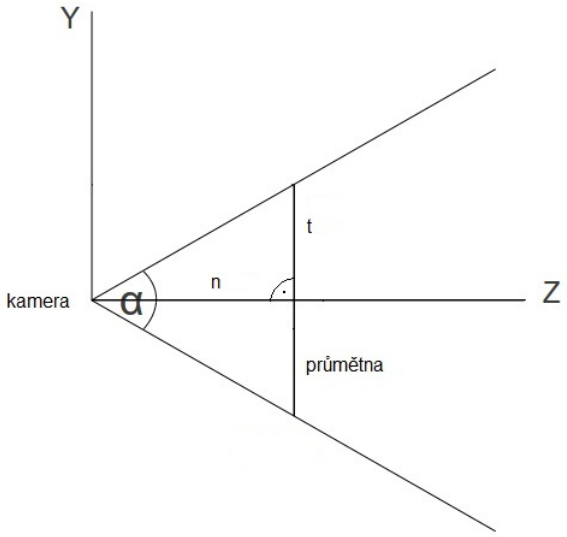
Kameru je potřeba posunout v opačném směru než je její pozice, aby se nacházela v počátku. Matice kamery má tvar:

$$\begin{pmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Všechny vrcholy modelu jsou poté vynásobeny maticí složenou z pohledové matice a matice kamery. Model je převeden do prostoru kamery, ten má počátek v místě, kde se nachází kamera. [1]

4.4 Perspektivní projekce

Perspektivní projekce neboli středové promítání simuluje vnímání lidského oka reálného světa. Při středovém promítaní dochází k zobrazení objektů na průmětnu. Předměty, které jsou dále od pozorovatele, se zobrazí menší než předměty blíže k pozorovateli. Perspektivní projekce je charakterizována středem promítání, tím je bod, ve kterém se setkají promítací paprsky procházející průmětnou, na kterou je zobrazen výsledný obraz. Zorné pole pozorovatele pak tvoří **zorný objem**, ten má tvar jehlanu a je tvořen šesti stěnami. Jehlan je převeden na krychli, jejíž vrcholy mají souřadnice v NDC (Normalized Device Coordinates) to znamená v intervalu <1,-1> a to z důvodu zjednodušení následného procesu ořezání.



Obrázek 3: Boční pohled na zorný objem

Pro sestrojení zorného objemu je potřeba znát zorný úhel kamery(α), poměr výšky a šířky výstupního zařízení($aspectRatio$), vzdálenost od průmětny (n) a vzdálenost od zadní stěny (f) tvořící zorný objem. Na základě těchto údajů lze spočítat jednotlivé hraniční souřadnice.

Hodnotu maximální souřadnice na ose Y (t), lze spočítat s využitím goniometrických funkcí, jak je možné vidět na obrázku 3.
$t = n \cdot tan(\frac{\alpha}{2})$

Hodnota minimální souřadnice na ose Y (b) je opačnou hodnotou hodnoty maximální souřadnice.
$b = -t$

S využitím poměru mezi výškou a šířkou výstupního zařízení je možné dopočítat hodnotu maximální souřadnice na ose X (r) s využitím hodnoty t.
$r = t \cdot aspectRatio$

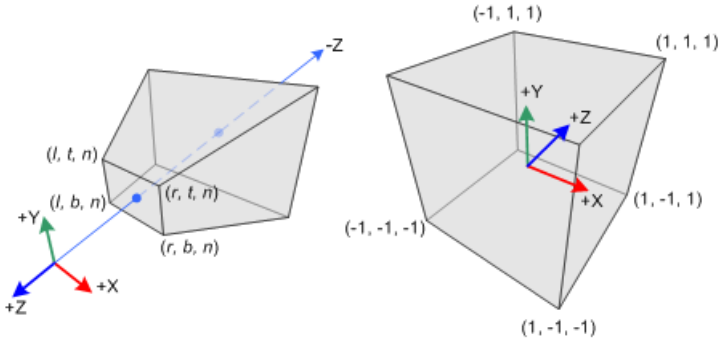
Hodnota minimální souřadnice na ose X (l) se pak dopočítá pomocí hodnoty l.
$l = -r$

Tyto hodnoty se dosadí do matice pro výpočet zorného objemu:

$$\begin{pmatrix} \frac{2*n}{(r-l)} & 0 & \frac{r+l}{(r-l)} & 0 \\ 0 & \frac{2*n}{(t-b)} & \frac{t+b}{(t-b)} & 0 \\ 0 & 0 & \frac{-(f+n)}{(f-n)} & \frac{-2*f*n}{(f-n)} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Výpočet jednotlivých elementů matice je odvozen na základě podobnosti trojúhelníků [5].

Ze zorného objemu získáme normalizovanou krychli pomocí perspektivního dělení, tedy vydělením všech souřadnic vrcholů homogenní souřadnicí $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1)$.



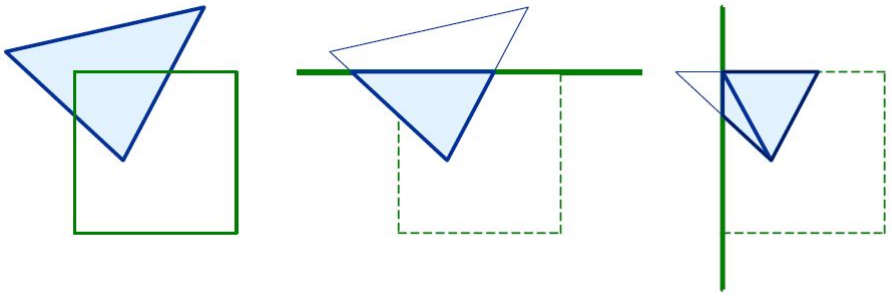
Obrázek 4: Převedení zorného jehlanu na jednotkovou krychli [1]

[1] http://www.songho.ca/opengl/gl_projectionmatrix.html

4.5 Ořezání zorným objemem

Při ořezání zorným objemem jsou z procesu rasterizace odstraněny objekty, případně jejich části, které se nacházejí mimo zorný objem. Při tomto procesu jsou všechny plochy aproximující povrch tělesa postupně ořezány rovinami ohraničující zorný objem. Těmito plochami jsou nejčastěji trojúhelníky a při jejich ořezání může dojít ke čtyřem případům.

1. Všechny vrcholy leží uvnitř zorného objemu - není prováděno ořezání.
2. Všechny vrcholy leží mimo zorný objem - není prováděno ořezávání.
3. Dva vrcholy leží mimo zorný objem - vytvoří se dva průniky s rovinou a z nich jeden nový trojúhelník.
4. Jeden vrchol leží mimo zorný objem - vytvoří se dva průniky s rovinou a dva nové trojúhelníky.



Obrázek 5: Ořezání trojúhelníka převzato z [1]

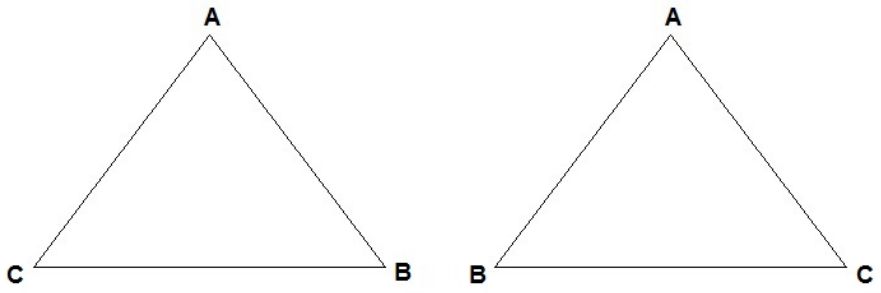
V případě, že koncové body některé hrany trojúhelníka leží na opačné straně řezové roviny, než se nachází zorný objem, je potřeba vypočítat průnik této hrany s řezovou rovinou. Přímku v prostoru lze zapsat parametricky jako $X = A + (B - A)\lambda$. Řezovou rovinu lze zapsat ve tvaru $X \cdot n = 0$, kde n je normálový vektor řezové roviny, například pro rovinu $x = 1$ je $n = (1, 0, 0, -1)$, pro rovinu $y = -1$ je $n = (0, -1, 0, -1)$ atd. Na základě znalostí těchto dvou rovnic lze vyjádřit parametr λ

$$\lambda = \frac{nA}{n(A - B)}, 0 \leq \lambda \leq 1$$

Po vypočtení se hodnota λ dosadí do parametrické rovnice přímky spolu s body určujícími přímku, výsledkem je průnik roviny a hrany trojúhelníka. Z těchto průniků a vrcholů trojúhelníka ležících uvnitř zorného objemu se sestaví jeden nebo dva nové trojúhelníky, které mohou být ořezány dalšími rovinami zorného objemu. [1]

4.6 Odstranění odvrácených ploch

Odstranění odvrácených ploch neboli Backface culling je metoda, která zajišťuje, že nebudou vykreslovány ty plochy, kterou jsou k pozorovateli natočeny rubovou stranou. Na tělesech se jedná o plochy nacházející se na zadních stěnách, tedy stěnách, které jsou odvráceny od pozorovatele. Při této metodě se využívá pořadí vrcholů v trojúhelníku. Pokud jsou seřazeny po směru hodinových ručiček, je trojúhelník natočen směrem k pozorovateli, v opačném případě je od pozorovatele odvrácen.



(a) K pozorovateli (b) Od pozorovatele

Obrázek 6: Orientace vrcholů trojúhelníka

V jakém pořadí jsou vrcholy v trojúhelníku lze snadno zjistit, a to na základě normálového vektoru trojúhelníka, pokud je jeho z-tová složka vetší než nula je trojúhelník natočen k pozorovateli přední stranou, v opačném případě zadní stranou. V prostoru je potřeba dvou směrových vektorů, provedeme mezi nimi vektorový součin a získáme tak vektor kolmý na tyto dva vektory, tedy vektor normálový. [4]

```
object3D.prototype.backFaceCulling = function() {

    for (var i = 0; i < this.triangles.length; i++) {
        var triangle = this.triangles[i];

        var p0 = new vector(triangle.p0.x, triangle.p0.y, triangle.p0.z);
        var p1 = new vector(triangle.p1.x, triangle.p1.y, triangle.p1.z);
        var p2 = new vector(triangle.p2.x, triangle.p2.y, triangle.p2.z);

        var U = p1.subtraction(p0);
        var V = p2.subtraction(p0);
        var normal = U.crossProduct(V);

        if (normal.z < 0)
            triangle.backFace = true;
        else {
            triangle.backFace = false;
        }
    }
}
```

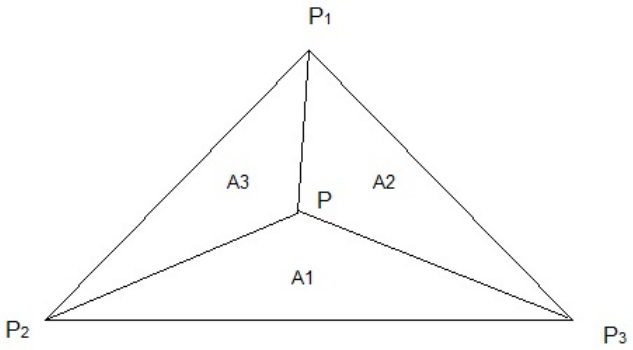
Výpis 6: Implementace funkce Backface culling

4.7 Mapování textur

Textura je vzorek, kterým může být pokryt povrch tělesa při procesu mapování textur. Je popsána dvourozměrným polem, které uchovává informace o barevných složkách jednotlivých bodů (texelů) textury. Při tomto procesu dochází k nanesení textury na povrch 3D modelu a je tak modelu dodán realistický vzhled. Textury nejčastěji představují nejrůznější materiály.

Jednotlivé body modelu si uchovávají odpovídající souřadnice textury, ty jsou označovány písmeny u, v a bývají zadávány v normalizovaném tvaru tedy v rozmezí <0,1>. Bodům je třeba souřadnice textury přiřadit nebo je dopočítat, což je potřeba po ořezávání zorným objemem při kterém vznikají nové body na povrchu tělesa. K dopočítání souřadnic textury se používají **barycentrické souřadnice**

Každý bod P uvnitř trojúhelníka, tvořeného vrcholy P_1, P_2, P_3, lze vyjádřit pomocí barycentrických souřadnic jako $P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3$ a také platí $\alpha + \beta + \gamma = 1$. Při výpočtu souřadnic textury se do rovnice dosazují souřadnice u nebo v jednotlivých bodů.



Obrázek 7: Rozdělení trojúhelníka na oblasti

Jednotlivé souřadnice α, β a γ lze spočítat jako poměr obsahu oblasti A1, A2, A3 a celého trojúhelníka.

$\alpha = \frac{A_1}{A_1+A_2+A_3}, \beta = \frac{A_2}{A_1+A_2+A_3}, \gamma = \frac{A_3}{A_1+A_2+A_3}$

Pro výpočet obsahu jednotlivých trojúhelníků lze využít například velikosti vektorového součinu. $S = \frac{1}{2}|u \times v|$. Tímto způsobem získáme barycentrické souřadnice bodu a s využitím znalostí souřadnic textury vrcholů trojúhelníka dopočítáme odpovídající souřadnice textury pro bod uvnitř trojúhelníka.[6]

Nanášení textury

Při nanášení textury je potřeba převést texturu z prostoru textury do prostoru scény, což je realizováno pomocí afinních transformací. Na základě znalostí souřadnic textury a souřadnic v prostoru scény je možné vypočítat jednotlivé prvky transformace. Objekty v elementu `canvas` lze transformovat pomocí metody `context.transform(a,b,c,d,e,f)`, protože pro vykreslování využíváme 2D `context`, také matice popisující transformaci je 2D homogenní maticí.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

V rovnici výše představují proměnné x', y' souřadnice pro vykreslení v elementu `canvas` a x, y představují souřadnice na textuře, kterou chceme vykreslit. Pro přehlednost místo označení x, y použiji u, v.

$x'_1 = a \cdot u_1 + c \cdot v_1 + e$
$y'_1 = b \cdot u_1 + d \cdot v_1 + f$

$x'_2 = a \cdot u_2 + c \cdot v_2 + e$
$y'_2 = b \cdot u_2 + d \cdot v_2 + f$

$x'_3 = a \cdot u_3 + c \cdot v_3 + e$
$y'_3 = b \cdot u_3 + d \cdot v_3 + f$

Pro jeden trojúhelník získáme šest rovnic o šesti neznámých. Jednotlivé prvky transformační matice pak lze vypočítat například pomocí Cramerova pravidla.

4.8 Transformace na výstupní zařízení

Po perspektivním dělení jsou souřadnice objektu v NDC tedy mezi <-1,1>, proto je potřeba souřadnice transformovat na souřadnice výstupního zařízení, to má tvar trojúhelníku o šířce w a výšce h s počátkem v bodě x, y. Výstupní zařízení představuje `canvas`, jehož počátek se nachází v levém horním rohu.

Vynásobením x-ové souřadnice daného vrcholu $\frac{w}{2}$ a y-ové souřadnice $\frac{h}{2}$ převedeme vrchol do souřadnic zařízení, do této doby byli souřadnice vrcholů v NDC. Přičtením $\frac{w}{2}$ a $\frac{h}{2}$ vycentrujeme scénu doprostřed zařízení. Pokud požadujeme určité odsazení, dosadíme za hodnoty x a y libovolné číslo představující odsazení v pixelech [16].

$x_w = \frac{w}{2} \cdot x_{ndc} + x + \frac{w}{2}$
$y_w = \frac{h}{2} \cdot y_{ndc} + y + \frac{h}{2}$

4.9 Rasterizace

Při rasterizaci dochází k vykreslování jednotlivých primitiv (trojúhelníků) na rastrové výstupní zařízení. V případě této práce rasterizaci zajišťují funkce elementu `canvas`.

Nejprve je vybrán obrys trojúhelníka pomocí funkcí pro vytváření cesty za využití souřadnic x_p, y_p, což jsou souřadnice trojúhelníka na výstupním zařízení. Funkce `beginPath`

začne vytváření nové cesty, pomocí funkcí `moveTo` a `lineTo` vytvoříme cestu a funkce `closePath` tuto cestu ukončí a spojí první bod cesty s bodem posledním. Metoda `clip` zajistí, že následné vykreslování metodou `drawImage` bude realizováno pouze v oblasti tvořené definovanou cestou. `Context` je však potřeba transformovat tak, aby vykreslená textura splňovala perspektivu. Transformaci `contextu` spočítáme s pomocí rovnic uvedených v kapitole 4.7, což zajišťuje implementovaná funkce `calculateTransformation`.

```
object3D.prototype.applyTexture = function(p0, p1, p2) {

    this.context.save();
    this.context.beginPath();
    this.context.moveTo(p0.xp, p0.yp);
    this.context.lineTo(p1.xp, p1.yp);
    this.context.lineTo(p2.xp, p2.yp);
    this.context.closePath();
    this.context.clip();

    var m = this.calculateTransformation(p0, p1, p2);

    this.context.setTransform(m[0], m[1], m[2], m[3], m[4], m[5]);
    this.context.drawImage(this.texture, 0, 0);
    this.context.restore();
}
```

Výpis 7: Vykreslení trojúhelníka s texturou v elementu canvas

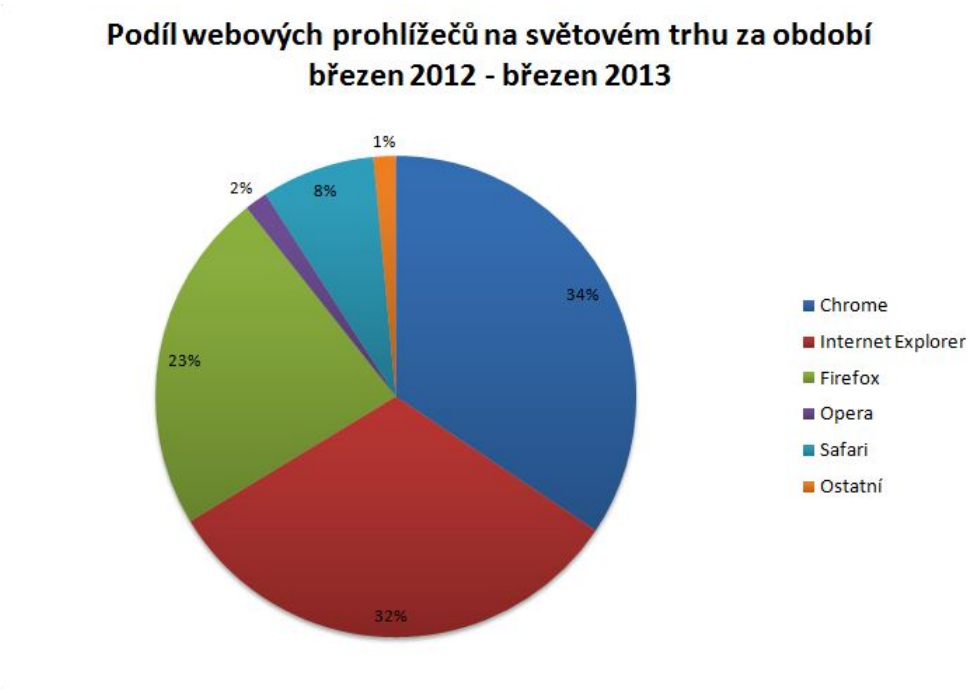
Na základě teoretických poznatků zmíněných v této kapitole jsem implementovala JavaScriptový renderer. Na obrázku 8 je ukázka vykreslení scény v elementu `canvas`, která obsahuje texturou potažený kvádr a kouli.

Obrázek 8: Vykreslení scény pomocí implementovaného rendereru

5 Webové prohlížeče

Webový prohlížeč slouží k zobrazování hypertextových dokumentů, ty jsou umístěny v celosvětové síti World Wide Web. Zobrazují HTML soubory, které mohou obsahovat obrázky, soubory PDF, videa a jiné. Webových prohlížečů existuje celá řada, v této práci se zabývám šesti nejpoužívanějším prohlížeči a to jak v desktopové verzi tak ve verzi mobilní.

Mezi nejpoužívanější prohlížeče patří Google Chrome, Mozilla Firefox, Internet Explorer, Opera, Safari a Maxthon. Na obrázku 9 je znázorněno využívání jednotlivých webových prohlížečů za poslední rok. Trhu vládne prohlížeč Chrome vyvíjený společností Google, následovaný Internet Explorerem od společnosti Microsoft, Firefox od firmy Mozilla je na třetím místě, s velkým odstupem jej následuje Safari od společnosti Apple, dále Opera vyvíjená firmou Opera Software, na posledním místě je Maxthon vyvíjený společností Maxthon International, který je zařazen mezi ostatními prohlížeči.

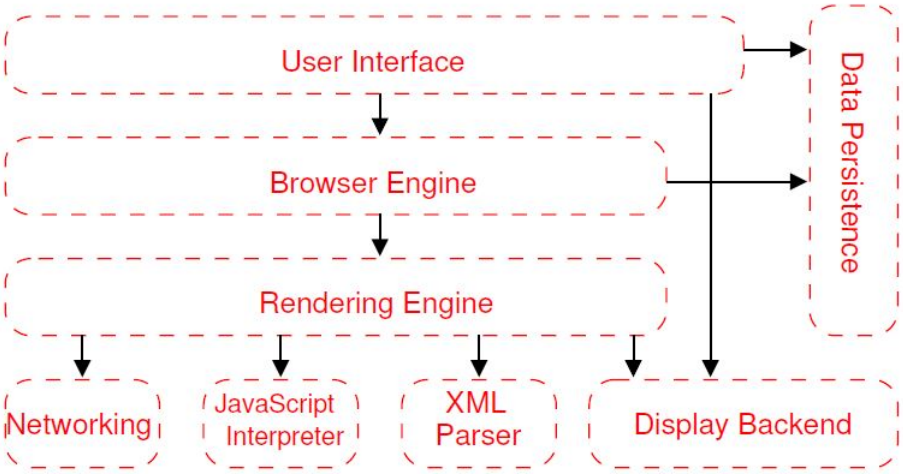


Obrázek 9: Podíl prohlížečů v období od března 2012 do března 2013 [2]

5.1 Architektura prohlížečů

Prohlížeče se skládají z osmi hlavních komponent,které navzájem spolupracují. Architektura je vyobrazena na obrázku 10.

[2]Zdroj: `http://gs.statcounter.com/`



Obrázek 10: Architektura webových prohlížečů

- User Interface (uživatelské rozhraní) je vrstva mezi uživatelem a prohlížečem, které poskytuje funkce jako adresní řádek, stahování obsahu, nastavené prohlížeče, tisk a podobné.
- Browser Engine (jádro prohlížeče) je komponenta, která zprostředkovává komunikaci mezi uživatelským rozhraním a renderovacím jádrem.
- Rendering Engine (renderovací jádro) zajišťuje vykreslování webové stránky a na základě použitých HTML tagů a CSS stylů vytváří renderovací strom, který poté vykreslí na monitor.
- Networking je komponenta, která je zodpovědná například za zasílání HTTP požadavků serveru.
- JavaScript Interpreter slouží k překladu kódu napsaného pomocí jazyka JavaScript, který může být ve stránce zahrnut.
- XML Parser slouží k parsování[3]XML elementů do objektů DOM.
- Display Backend zobrazuje prvky uživatelského rozhraní jako tlačítka nebo "zaškrtávací políčka"a mnoho dalších.
- Data Persistance (dočasná data) slouží k ukládání dat na lokální disk, kde je prohlížeč nainstalován, například záložky, cookies nebo bezpečnostní certifikáty.

[3] při parsování dochází k získávání jednotlivých hodnot z řetězce znaků

5.2 Renderovací jádra prohlížečů

Renderovací jádro zajišťuje zobrazení obsahu ve webovém prohlížeči. Jedná se o software, který načítá zdrojové soubory HTML a CSS společně s obrázky a vykresluje výsledný naformátovaný obsah do prohlížeče.

To, jakým způsobem bude obsah interpretován, závisí na webovém prohlížeči, který zdrojový kód zpracovává. Různé webové prohlížeče využívají různá jádra, proto se může stát, že při zobrazení téže webové stránky v různých prohlížečích získáme různý výsledek. Toto je částečně ošetřeno pomocí definovaných standardů, kterými by se prohlížeče měli řídit.

Tabulka 1: Přehled prohlížečů a jejich jader

Prohlížeč	Renderovací jádro	JavaScriptové jádro
Google Chrome	WebKit	V8
Mozilla Firefox	Gecko	SpiderMonkey
Internet Explorer	Trident	Chakra
Opera	Presto	Carakan
Safari	WebKit	SquirrelFish
Maxthon 4	WebKit, Trident	V8

V tabulce 1 je vypsán seznam šesti nejpoužívanějších prohlížečů, jejich renderovacích jader a JavaScriptových jader. Zajímavý je zvláště prohlížeč Maxthon, který využívá jak jádro WebKit, tak Trident. V základu je nastaven WebKit, který je lepší při vykreslování moderních stránek. V případě prohlížení starších stránek Maxthon přepne na jádro Trident, které je pro zobrazení těchto stránek lepší.

5.2.1 WebKit

WebKit je open source [4]renderovací jádro, vyvíjené společnosti Apple napsané v jazyce C++. Využívají jej prohlížeče Safari, Chrome, Maxthon a další. Dále je využíván ve webovém prohlížeči elektronické čtečky Amazon Kindle. Je také využíván v jiných aplikacích jako například emailový klient Mail společnosti Apple využívaný v OS X.

Je postaven na KHTML a KJS vyvíjených v rámci projektu KDE. KHTML je renderovací jádro ve Webkitu nazývané WebCore a KJS je interpret jazyka JavaScript ve Webkitu nazávaný jako JavaScriptCore byl později přepracován na SquirrelFish (označován i jako Nitro). Google Chrome však využívá pouze WebCore a k interpretaci JavaScriptových kódů používá jádro V8.

Webkit je součástí nejrozšířenějšího webového prohlížeče a je tak nejpoužívanějším jádrem s více než 40% na celosvětovém trhu.

[4]software s volně dostupným zdrojovým kódem

5.2.2 Trident

Trident je druhým nejpoužívanějším renderovacím jádrem, které je vyvíjeno společností Microsoft a je využíván v prohlížeči Internet Exlorer, Maxthon a mnoha dalších. Je také označován jako MSHTML a poprvé byl použit v Internetu Explorer 4.0.

Internet Exlorer využívá architekturu komponent, jednou z těchto komponent je právě Trident, který ma za úkol vykreslování obsahu uvnitř webového prohlížeče. Je také využíván i jiných aplikacích, kterými jsou například Visual Studio od firmy Microsoft nebo Microsoft Outlook. V internetu Explorer je JavaScriptové jádro označováno jako Chakra.

5.2.3 Gecko

Gecko je renderovací jádro vyvíjené společností Mozilla, původně bylo označováno jako NGLayout. Je napsáno v jazyce C++ a je multiplatformní. Je využíván ve webovém prohlížeči Firefox , který má podíl 23% na trhu webových prohlížečů a je třetím nejvyužívanějším prohlížečem na světe za prohlížeči Google Chrome a Internet Explorer. Verze jádra Gecko jsou číslovány stejně jako verze prohlížeče Firefox, aktuální verzí v době psaní tohoto textu je 20.1.

5.2.4 Presto

Presto je renderovací jádro vyvinuté společností Opera Software a používané v prohlížečích Opera a to do verze 12.15. Ve verzích následujících bude využíváno jádro WebKit. JavaScriptové jádro Opery je označováno jako Carakan. Na rozdíl od ostaních renderovacích jader se nejedná o open source projekt. Jeho kód tedy nebyl přístupný pro vývojáře, což by se mohlo změnit ukončením vývoje tohoto jádra.

6 Testování výkonnosti prohlížečů

Výkonnost webových prohlížečů byla měřena pomocí vykreslování scén v implementovaném rendereru. K měření FPS bylo využito monitorovacího nástroje stats.js [5].

Při měření výkonnosti byl do scény nejprve umístěn nejjednodušší kvádr, jeho složitost byla postupně zvyšována a byli přidávány objekty koule. V poslední scéně se nachází 15 koulí. Složitost scén je vyjádřena pomocí trojúhelníků aproximujících povrch těles ve scéně a je zobrazena v tabulce 2.

Měření lze rozdělit do třech částí, nejprve byly měřeny výkonnosti aktuálních verzích na třech různých počítačích, poté jsem se zaměřila na srovnání výkonnosti mezi novými verzemi a staršími na jednom počítači. Poslední část je věnována výkonnosti v mobilních zařízeních.

Tabulka 2: Složitost jednotlivých scén

Scéna	Složitost
1	12
2	48
3	108
4	364
5	472
6	768
7	1280
8	2048
9	2560
10	3840

6.1 Výkonnost desktopových prohlížečů

Prohlížeče byli testovány na třech sestavách o různých hardwarových parametrech, dvou noteboocích a stolním počítači. Na jeden s notebooků nebylo možné nainstalovat prohlížeč Safari, proto se údaje z toho prohlížeče v grafu neobjevují.

Měření na první sestavě

První sestavou je notebook Acer Emachines e725, jedná se o slabší přístroj s dvou jádrovým procesorem a integrovanou grafickou kartou. Parametry toho počítače jsou vypsány v tabulce 3.

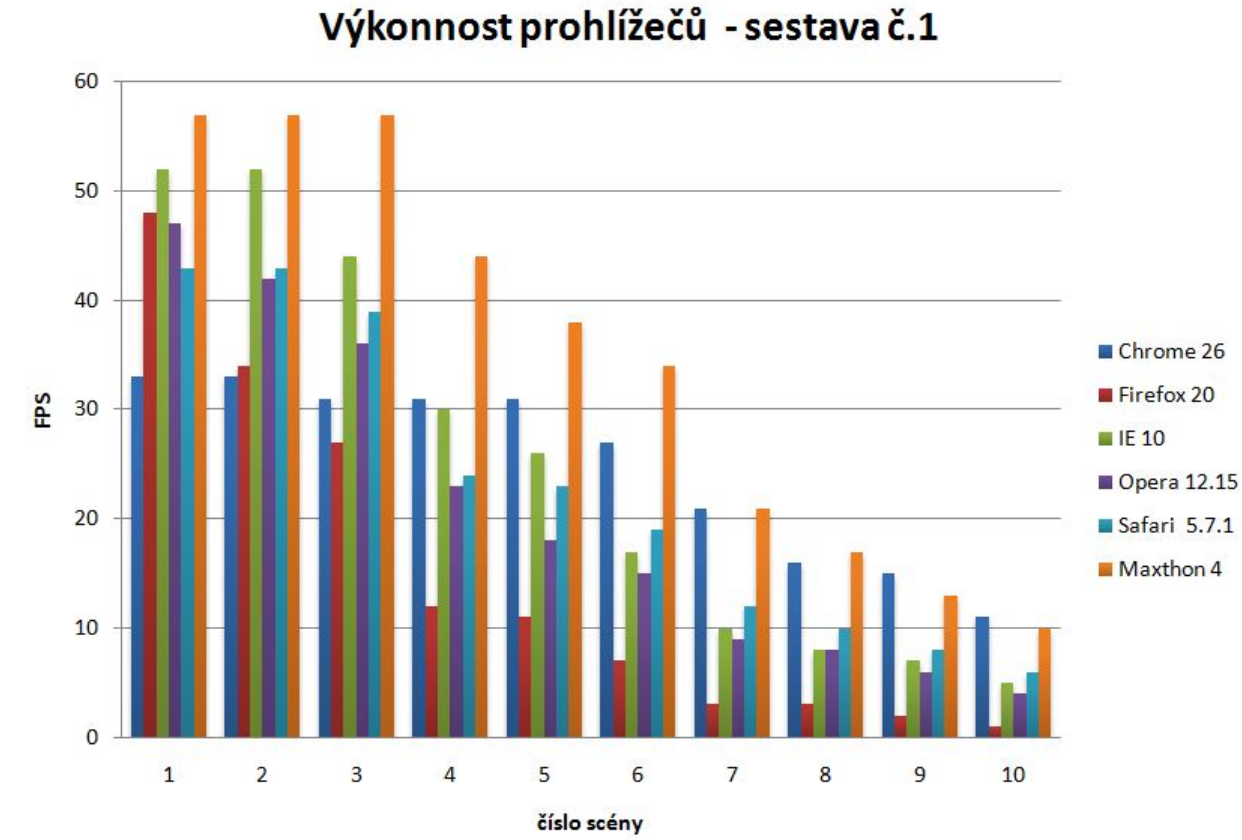
Výsledky měření je možné vidět v grafu na obrázku 11. V prvních scénách byl výrazně nejrychlejší Maxthon, po něm Internet Exlorer a ostaní prohlížeče. V nejnáročnější scéně byl nejvýkonnější Chrome, po něm Maxthon, Safari, Internet Explorer, Opera a jako

[5] Zdroj: `https://github.com/mrdoob/stats.js/`

Tabulka 3: Technické parametry testovací sestavy č.1

CPU	Intel Pentium (R) Dual-Core T4300 2,1 GHz
RAM	3 GB
Grafická karta	Intel GMA 4500MHD
OS	Windows 7 Professional 32bit Service Pack 1

poslední Firefox. Nejhůř dopadl Firefox, který v náročných scénách podstatně zaostával za ostatními prohlížeči.



Obrázek 11: Výkonnost prohlížečů - sestava č.1

Měření na druhé sestavě

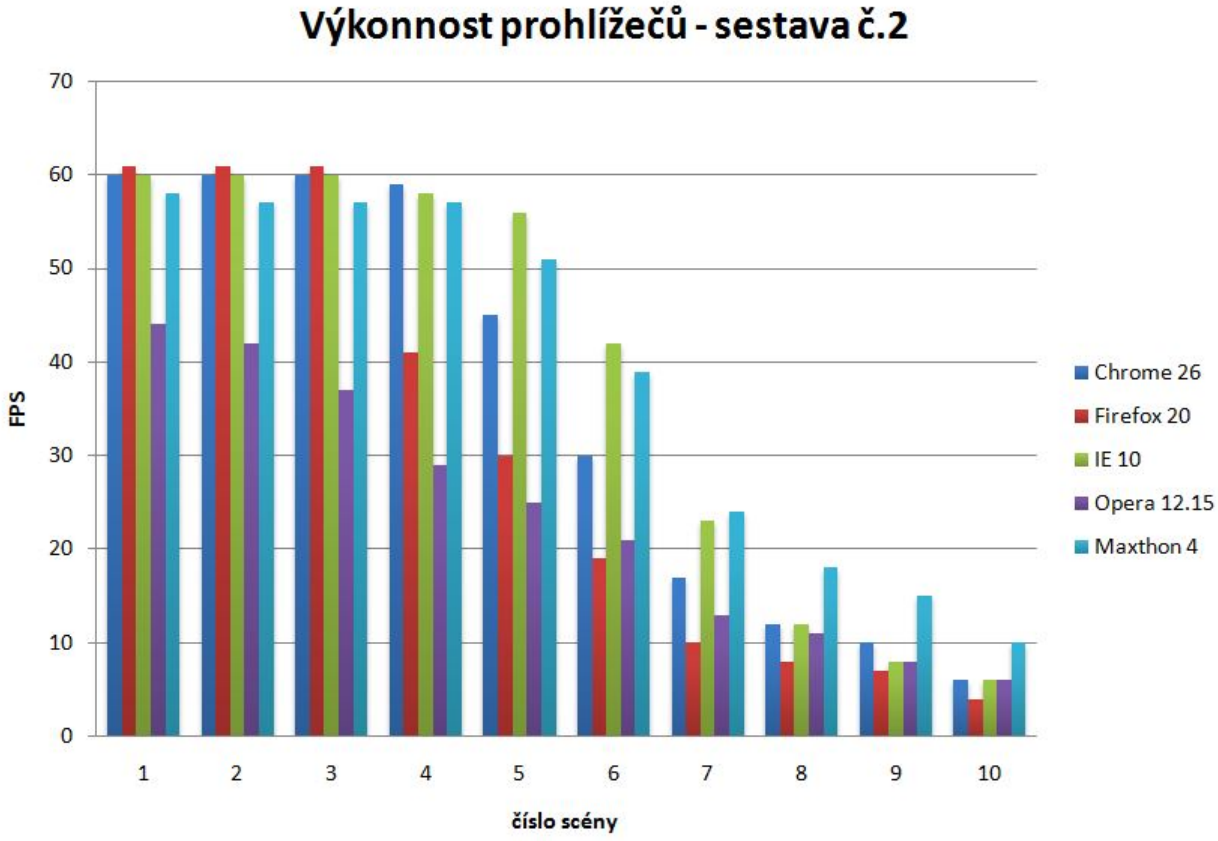
Druhá sestava je výkonnější, jedná se o notebook GIGABYTE Q1580P-934PCZ s dvou jádrovým procesorem a grafickou kartou NVIDIA.

Graf s výsledky je možné vidět na obrázku 12. Při měření na této sestavě se Google Chrome choval u prvních scén výkonněji než u předchozí sestavy, avšak v náročnějších

Tabulka 4: Technické parametry testovací sestavy č.2

CPU	Intel(R) Core(TM) 2 DUO P8700 @2,53 GHz
RAM	4 GB
Grafická karta	NVIDIA GeForce GT 130M
OS	Windows 8 Pro Professional 64bit

scénách byl pomalejší. Ostatní prohlížeče se u náročnějších scén chovali výkonněji než u předchozí sestavy. Nejlépe si s náročnými scénami poradil Maxthon, hned za ním stejně dopadli Internet Explorer, Chrome, Opera a nejméně výkonný byl opět Firefox, který ze začátku sice výkonnostně stačil ostatním, ale se zvyšováním náročnosti scén se jeho výkonnost rapidně snížila.



Obrázek 12: Výkonnost prohlížečů - sestava č.2

Měření na třetí sestavě

Třetí sestava je nejvýkonnější, jedná se o stolní počítač s parametry uvedenými v tabulce 5.

Výsledky z této sestavy byli velmi podobné jako z té předchozí, nacházejí se v grafu na obrázku 13. V náročných scénách byly naměřeny nepatrně vyšší výsledky. Oproti předchozí sestavě byl do měření zahrnut také prohlížeč Safari. Opět byl v nejnáročnějších scénách nejvýkonnějším prohlížečem Maxthon, po něm Chrome a ostatní prohlížeče. Firefox se opět ukázal jako prohlížeč s nejslabším renderovacím jádrem, sice v prvních scénách držel krok s ostatním prohlížeči a byl nejprve lepší než Opera, Safari a Maxthon, ale s přibývající složitostí scén klesala jeho výkonnost více než ostatních prohlížečů.

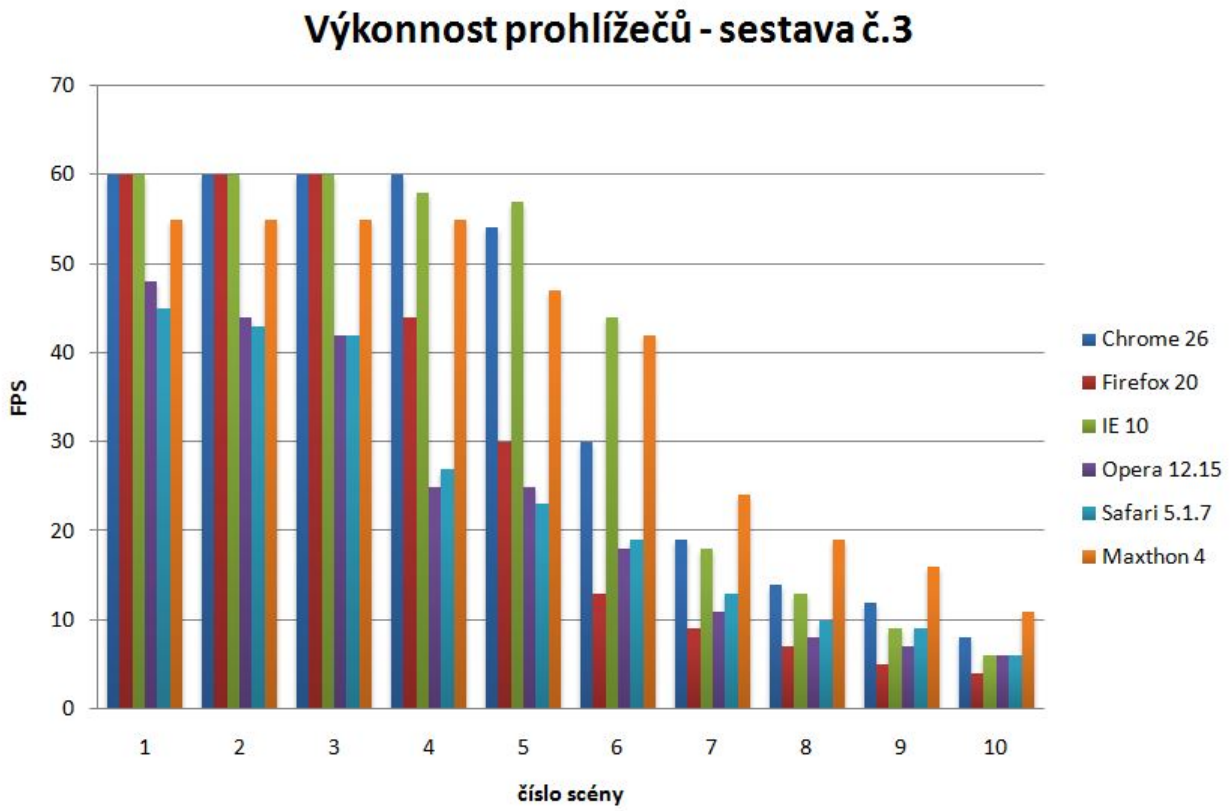
Tabulka 5: Technické parametry testovací sestavy č.3

CPU	Intel Pentium Dual-Core E6600 @3,06 GHz
RAM	4 GB
Grafická karta	NVIDIA GeForce GTX 460 SE
OS	Windows 7 Ultimate 64bit

Zhodnocení

Na základě získaných výsledků bych jako nejvýkonnější jádro označila Webkit, který je obsažen v prohlížečích Chrome, Safari a Maxthon. Za ním je jádro Trident, na tomto jádře běží prohlížeče Internet Explorer a Maxthon. Třetím je Presto v prohlížeči Opera. Nejhorší bylo jádro Gecko, které se používá k vykreslování v prohlížeči Firefox.

Rozdíly v jednotlivých měřeních jsou způsobeny jak hardwarovými parametry jednotlivých sestav, tak i různými operačními systémy. U prohlížeče Chrome je rozdíl výkonnosti na slabší a výkonnější sestavě také způsoben jeho funkcí hardwarové akcelerace `canvasu`.



Obrázek 13: Výkonnost prohlížečů - sestava č.3

6.2 Porovnání výkonnosti různých verzí prohlížečů

Na trhu s webovými prohlížeči dochází k boji o nejvýkonnější software a proto jsou jednotlivé verze prohlížečů čím dál tím rychlejší a nabízejí mnohem větší podporu nejrůznějších funkcí.

Některé společnosti vydávají stabilní verze častěji jako například Google nebo Mozilla a mezi těmito verzemi není zvlášť velký rozdíl. Pro ilustraci vývoje výkonnosti jsem si zvolila verze, které byly vydány s delším rozestupem. Porovnávám vždy verze aktuální v době psaní tohoto textu (duben 2013) a verzi starší. Výsledky měření se nacházejí v příloze v grafu na obrázku 15.

Google Chrome

Výkonnost prohlížeče Google Chrome byla měřena ve verzi 26, která byla vydána 28.3.2013 a ve verzi 17 vydané 8.2.2012. Z grafu je patrné různé chování prohlížeče v těchto dvou verzích. Verze starší je v méně náročných scénách výkonnější, kdežto verze novější je výkonnější ve scénách náročných. Chrome 26 má oproti verzi 17 mnoho vylepšení, jedním z nich je hardwarová akcelerace elementu `canvas`, což může způsobovat rozdíly ve výkonnosti jak v lepší výkonnosti ve složitějších scénách, tak v horší výkonnosti ve scénách slabších, protože tato akcelerace nemusí vždy fungovat úplně správně. Verze 26 obsahuje také oproti verzi 17 novější verzi JavaScriptového jádra V8.

Mozilla Firefox

Výkonnost prohlížeče Mozilla Firefox byla měřena ve verzi 20 vydané 11.4.2013 a ve verzi 15 vydané 6.9.2012. I když je mezi těmito verzemi téměř půl roční rozdíl nejsou patrné nějak zásadní změny ve výkonnosti. Novější verze je v jednodušších scénách nepatrně výkonnější, ale ve scénách složitějších se výkonnost nelišila. Verze 20 obsahuje rychlejší JavaScriptový překladač IonMonkey.

Internet Explorer

Dalšími porovnávaným prohlížečem je Internet Explorer a to ve verzi 10, která byla vydána 26.10.2012 pro Windows 8 a pro Windows 7 SP1 26.3.2013 a ve verzi 9 vydané 14.3.2011. Opět je podle očekávání verze novější rychlejší a to po celou dobu testování.

Safari

Největší zaznamenaný rozdíl je mezi verzemi Safari 4.0 z 12.5.2009 a 5.1.7 vydané 9.5.2012. Výkonnost verze 5.1.7 je mnohonásobně vyšší než výkonnost verze 4.0. Což je způsobeno velkým časovým rozestupem mezi vydáním testovaných verzí. Protože se společnost Apple zaměřuje především na vývoj pro MAC OS a iOS, nejsou vydávány tak často nové verze prohlížeče Safari pro operační systém Windows.

Opera

Opera byla testována ve verzi 12.15 ze 4.4.2013 a ve verzi 11 vydané 16.12.2010. Mezi těmito verzemi je celkem velký časový odstup, což však nemělo zvlášť velký vliv na výkonnost. Novější verze je jen nepatrně rychlejší než verze starší, dalo by se však očekávat, že při takto velkém odstupu jednotlivých verzí bude rozdíl mnohem značnější.

Maxthon

Výkonnost tohoto prohlížeče byla měřena na verzi 3 vydané na podzim roku 2010 a na verzi 4 z 10.12.2012. Starší verze se chovala nejprve rychleji než verze novější. Ve složitějších scénách byla o něco pomalejší.

Zhodnocení

Jak je možné vidět z měření na jednotlivých prohlížečích, jejich výkonnost s každou novou vydanou verzí roste. Přibývají nové funkce, a to i funkce pro optimalizaci výkonu. Takovou je hardwarová akcelerace u Chromu, která se na různých počítačích chová různě a může způsobovat zhoršení výkonnosti prohlížeče.

Výkonnost prohlížečů ve velmi náročných scénách byla velmi malá, pro složitější vizualizace nepoužitelná. Ve scénách méně náročných se prohlížeče chovali dostatečně výkonně.

6.3 Výkonnost mobilních prohlížečů

Výkonnost mobilních verzí prohlížečů je mnohonásobně nižší z důvodu menší výkonnosti mobilních zařízení. Bohužel není možné na jedno zařízení nainstalovat všechny potřebné prohlížeče, tak jako je to možné u počítačů s operačním systémem Windows 7. Proto jsem tato měření prováděla na pro mě dostupném zařízení, na které bylo možné nainstalovat nejvíce prohlížečů. Tímto zařízení je tablet Asus Nexus 7.

Měření na zařízení Nexus 7

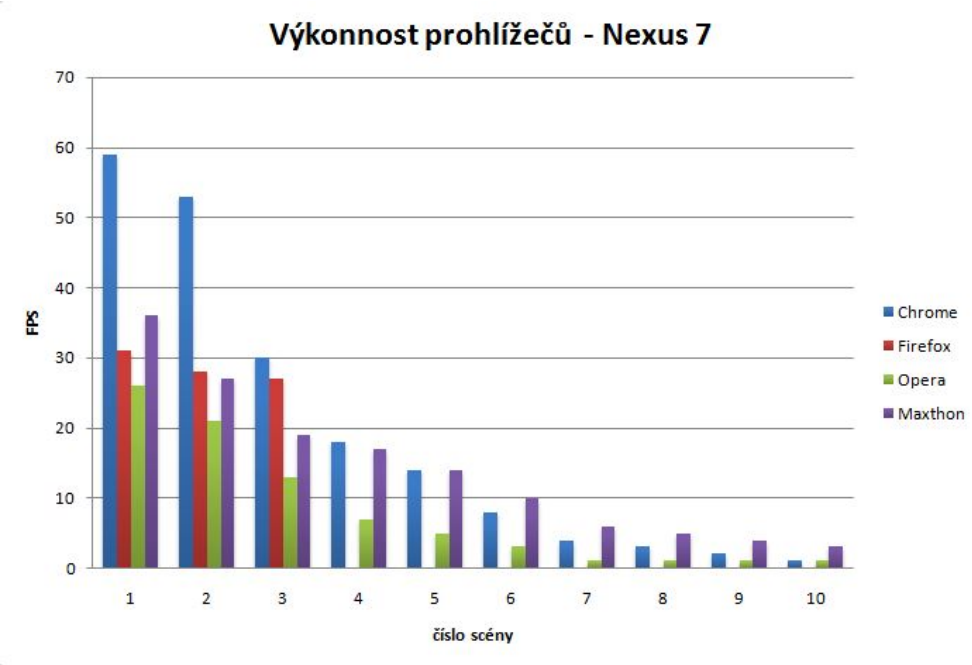
Výkonnost na tomto zařízení není možné srovnávat s výkonnosti předchozích počítačů, ale je zde možné vidět, že zobrazování jednoduchých scén v elementu `canvas` je možné i v mobilní prohlížečích. Technické parametry zařízení Nexus 7 se nacházejí v tabulce 6.

Tabulka 6: Technické parametry zařízení Nexus 7

Chipset	NVIDIA Tegra 3 T30L Quad-Core @1,2 GHz 12 grafických jader
RAM	1 GB
OS	Android 4.2.2

Graf s naměřenými hodnotami FPS je možné vidět na obrázku 14. Jako nejvýkonnější se nejprve projevoval Chrome, po něm Maxthon, Firefox a Opera. Nejsložitější scény však

nejlépe zvládal Maxthon za ním Chrome a Opera. Při vykreslování scén ve Firefoxu se vyskytli problémy, od scény, ve které byla přidána koule, prohlížeč přestal vykreslovat do elementu `canvas`.



Obrázek 14: Výkonnost prohlížečů - Nexus 7

7 Závěr

Výsledkem této práce je implementovaný renderer pro vykreslování scén ve webovém prohlížeči, který ukazuje možností elementu `canvas` technologie HTML5. Při vytváření tohoto rendereru jsem získala mnoho znalostí z oblasti počítačové grafiky, které jsem následně prakticky aplikovala. Implementace zahrnuje umístění kamery do scény, perspektivní projekci, odstranění odvrácených ploch, ořezání zorným objemem, mapování textur a rasterizaci.

Renderer byl využit k měření výkonnosti webových prohlížečů, které jsou schopné vykreslovat průměrně náročné scény, u těch velmi složitých je jejich výkonnost podstatně nižší až téměř nepoužitelná.

Renderer je možné využít k vytváření různých simulací ve webovém prohlížeči, pro zobrazování virtuálních prohlídek. Je však potřeba jej rozšířit. V přiložené dokumentaci jsou popsány jednotlivé funkce, které je možné využít pro vytváření vlastních scén.

Práci na rendereru se plánuji dále věnovat a rozšířit jej například o řešení viditelnosti při překrývání jednotlivých objektů, zavést do scény zdroj světla pro dokreslení reálnosti scén, dále pokrytí povrchu objektů jednobarevnými plochami a stínování těchto ploch.

8 Reference

[1] SOJKA, Eduard. *Počítačová grafika II: průvodce studiem*. Ostrava: Vysoká škola báňská - Technická univerzita, Fakulta elektrotechniky a informatiky, 2003, 25 l. ISBN 80-248-0293-7.

[2] LUBBERS, Peter, Brian ALBERS a Frank SALIM. *HTML5: programujeme moderní webové aplikace*. Vyd. 1. Brno: Computer Press, 2011, 304 s. ISBN 978-80-251-3539-6.

[3] ŽÁRA, Jiří. *Moderní počítačová grafika*. Vyd 1. Brno: Computer Press, 2004, 609 s. ISBN 80-251-0454-0.

[4] LAMBERTA, Billy. *Foundation HTML5 animation with javascript*. New Edition. [New York] : Friends of Ed: Apress, 2011. ISBN 978-143-0236-658.

[5] LENGYEL, E. *Mathematics for 3D: game programming and computer graphics*. Vyd. 1. Massachusetts: Charles River Media, 2004, s. 121-124. ISBN 15-845-0277-0.

[6] FLETCHER DUNN, Ian Parberry. *3D math primer for graphics and game development*. 2nd ed. Boca Raton, FL: A K Peters/CRC Press, 2011, s. 321-329. ISBN 978-1-4398-6981-9.

[7] GROSSKURTH, Alan a Michael W. GODFREY. *Architecture and evolution of the modern web browser* [online]. 2006 [cit. 2013-04-15].

[8] X3D. What is X3D?. *Web3D Consortium* [online]. © 1999-2013 [cit. 2013-04-1]. Dostupné z: http://www.web3d.org/realtime-3d/x3d/what-x3d/

[9] Java3D. *Oracle* [online]. [cit. 2013-04-01]. Dostupné z: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html

[10] WebGL Getting Started. *WebGL Wiki* [online]. 2011 [cit. 2013-04-1]. Dostupné z: http://www.khronos.org/webgl/wiki/Getting_Started

[11] VRML 1.0 Specification. *Web3D Consortium* [online]. © 1999-2013 [cit. 2013-04-1]. Dostupné z: http://www.web3d.org/x3d/specifications/vrml/VRML1.0/index.html

[12] OpenGL Projection Matrix. *OpenGL* [online]. © 2008-2012 [cit. 2013-04-08]. Dostupné z: http://www.songho.ca/opengl/gl_projectionmatrix.html

[13] *The WebKit Open Source Project* [online]. [cit. 2013-04-15]. Dostupné z: http://www.webkit.org/

[14] *Gecko* Gecko. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-15]. Dostupné z: http://en.wikipedia.org/wiki/Gecko_(layout_engine)

[15] Internet Explorer Architecture. *MSDN* [online]. © 2013 [cit. 2013-04-15]. Dostupné z: http://msdn.microsoft.com/en-us/library/aa741312(v=vs.85).aspx

[16] OpenGL Transformation. *OpenGL* [online]. © 2008-2013 [cit. 2013-04-23]. Dostupné z: http://www.songho.ca/opengl/gl_transform.html

A Obsah přiloženého CD

Na přiloženém CD se nachází praktická část této bakalářské práce, kterou je renderer napsaný v jazyce JavaScript. Organizace složek na CD je následující:

- **SRC** - složka se zdrojovými soubory
  - scripts - složka obsahující jednotlivé skripty tvořící renderer
  - `index.html` - ukázková scéna
  - `scene_1.html – scene_10.html` - scény využité k testování výkonnosti
- **Text** - text bakalářské práce ve formátu PDF-A
- **Dokumentace** - složka obsahuje HTML soubory s vygenerovanou dokumentací všech skriptů, pro zobrazení dokumentace je potřeba otevřít soubor `index.html`

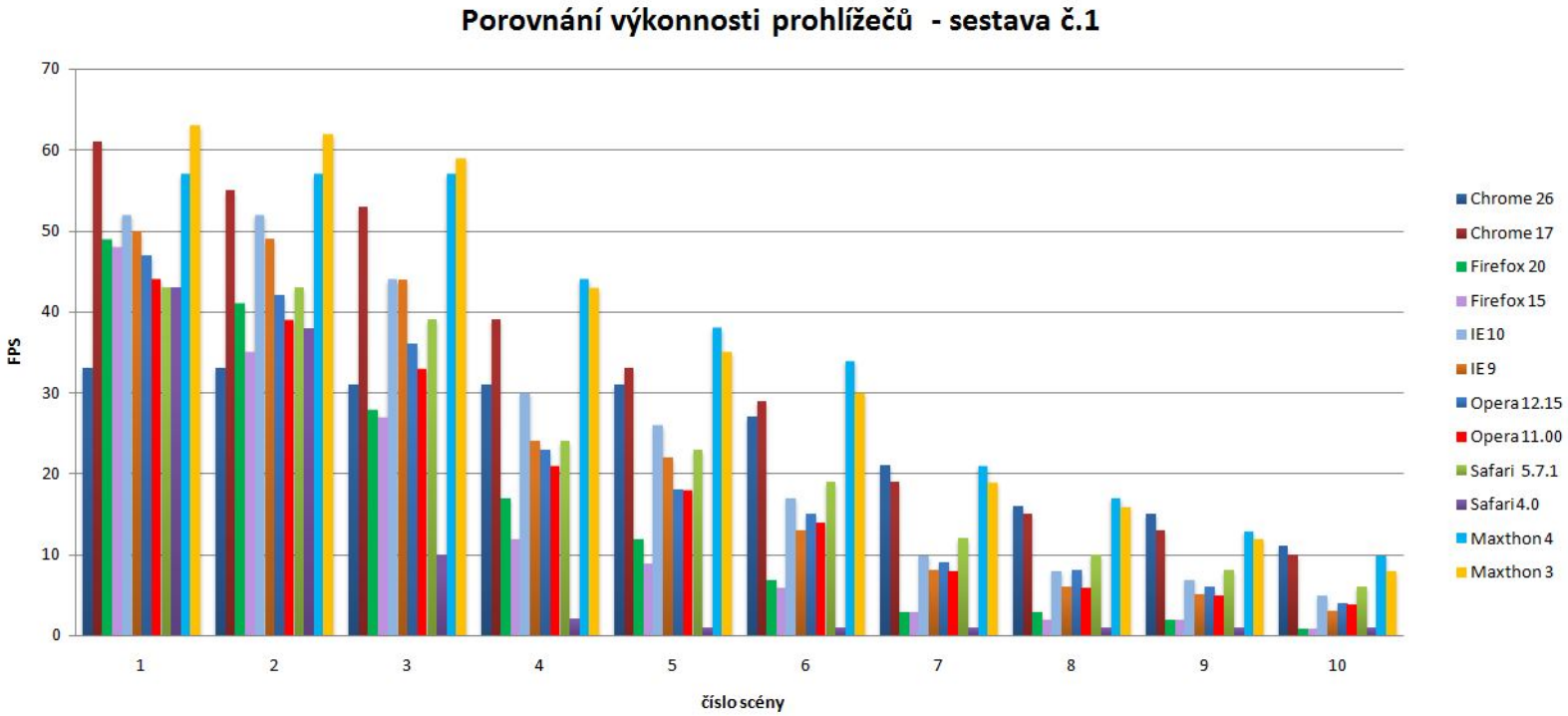
B Použití rendereru

Pro vykreslování pomocí rendereru je potřeba vložit do HTML stránky jednotlivé skripty `<script src="scripts/object3d.js"></script>` a provést následující kroky.

- **Kamera** představuje pozici pozorovatele, přidá se do scény vytvořením nové instance kamery `myCam = new camera(ctx)`, kde `ctx` představuje context canvasu, na který se bude vykreslovat.
- **Scéna** se vytvoří pomocí příkazu `myScene = new scene(myCam)`. Scéně je potřeba předat jako parametr kameru. Jednotlivé objekty se do scény přidávají jako prvky pole `objects`, tedy `myScene.objects.push(myObject)`.
- **Objekty** je možné vytvořit vlastní pomocí konstruktoru `myObject = new object3D()`, takovému objektu je potřeba přiřadit pole bodů na jejich povrchu `myObject.vertices = []`, přiřadit jednotlivé body k plochám pokrývajícím povrch `myObject.faces=[[0, 1, 2, 3], [2, 3, 5, 6], ...]`, pozici ve scéně `myObject.position = new vector(x, y, z)`, definovat počet bodů na výšku ploch `myObject.levelH = 1`, počet bodů na šířku ploch `myObject.levelV = 1` a přiřadit texturu pomocí `myObject.loadTexture(path)`.

  Nebo je možné využít již předdefinované objekty, ty se nacházejí ve skriptu `models.js` a jsou jimi kvádr a koule s různou úrovní detailů. Takové objekty se vytváří `mySphere = new sphere_H(x, y, z)`, kde x, y, z představují pozici ve scéně, pak už jen stačí definovat texturu stejným způsobem jako u modelů vlastních.

C Graf



Obrázek 15: Porovnání výkonnosti různých verzí prohlížečů